Metadata Management – Building Blocks

Abstract

As businesses strive to meet the regulatory and internal demands for strong governance and data lineage there is a necessity to gain as much value as possible from the information they have. For those using SAS Data Integration Studio there is often a wealth of information stored in metadata that can be of significant value if properly harnessed. This is the first paper in the Metadata Management series and it focuses on the concepts and building blocks required to begin managing metadata.

Introduction

Metadata is typically defined as "a set of data that describes and gives information about other data". Within SAS Data Integration Studio, as developers create and enhance jobs they are constantly generating metadata, and this metadata should be validated in line with development standards. This paper will first cover the terminology and concepts you need to begin to understand metadata management. After establishing the necessary building blocks there will be some worked examples that show how to use the tools provided. Finally everything that has been discussed will be brought together to provide some general solutions to the following problems:

- Are there any User Written nodes within a Job?
- Do multiple metadata table objects reference the same physical table?
- Has a particular property been set to the desired value?
- Does a flow begin and end with some generic start and stop jobs?

In order to be able to answer these questions we first need to have the tools to harness metadata.

Metadata – Terminology

The first step in metadata management is understanding the following terminology:

- Metadata types – The various groups of metadata objects e.g. Job, PhysicalTable, Tree
- Metadata repository – A collection of related metadata objects (the default repository is Foundation)
- Metadata attributes – Information about a specific metadata object e.g. name, description, create date, modified date
- Metadata associations – Information about which metadata objects are linked to other metadata objects e.g. a job can be linked to a deployed job and is always linked to a tree folder in the metadata tree
- Metadata properties – Information about the properties of a metadata object e.g. the APPENDFORCE or COLMACROVARS properties
- Metadata identifier – An ID for a metadata object in the format "XXXXXXXX.XXXXXXXX" where "X" can be character or numeric e.g. "A1BCD2EF.GH123IJ4"
- OMSOBJ – Stands for "Ontology Management Studio Object" and is used at the start of URIs.
- Uniform Resource Identifier (URI) – The combination of a metadata type with a metadata identifier frontloaded with OMSOBJ e.g. "OMSOBJ:JOB\A1BCD2EF.GH123IJ4"

Metadata – Concepts

The second step in metadata management is understanding the following concepts:

- All metadata objects are connected together via the metadata tree
- Within the metadata tree there are many branches, but it is all connected. This means you can navigate from any point to any other point within the metadata tree
- In order to gain information about a particular metadata object we need to look at its attributes or properties
- In order to traverse across the metadata tree we need to look at an objects associations and move from one metadata object to the next
- Associations always come in pairs. I.e. if a Job is associated to a Deployed Job, then that Deployed Job will also be associated back to the Job
- When you are issuing metadata commands you will be looking at the contents of one metadata repository at a time.

Figure 1 shows an example of the types of association you might find between metadata objects. In the example there is one parent tree folder, with four sub-tree folders that contain SAS Library, Physical Table, Job, and Deployed Job objects respectively. These objects have further associations and are interconnected in various ways.

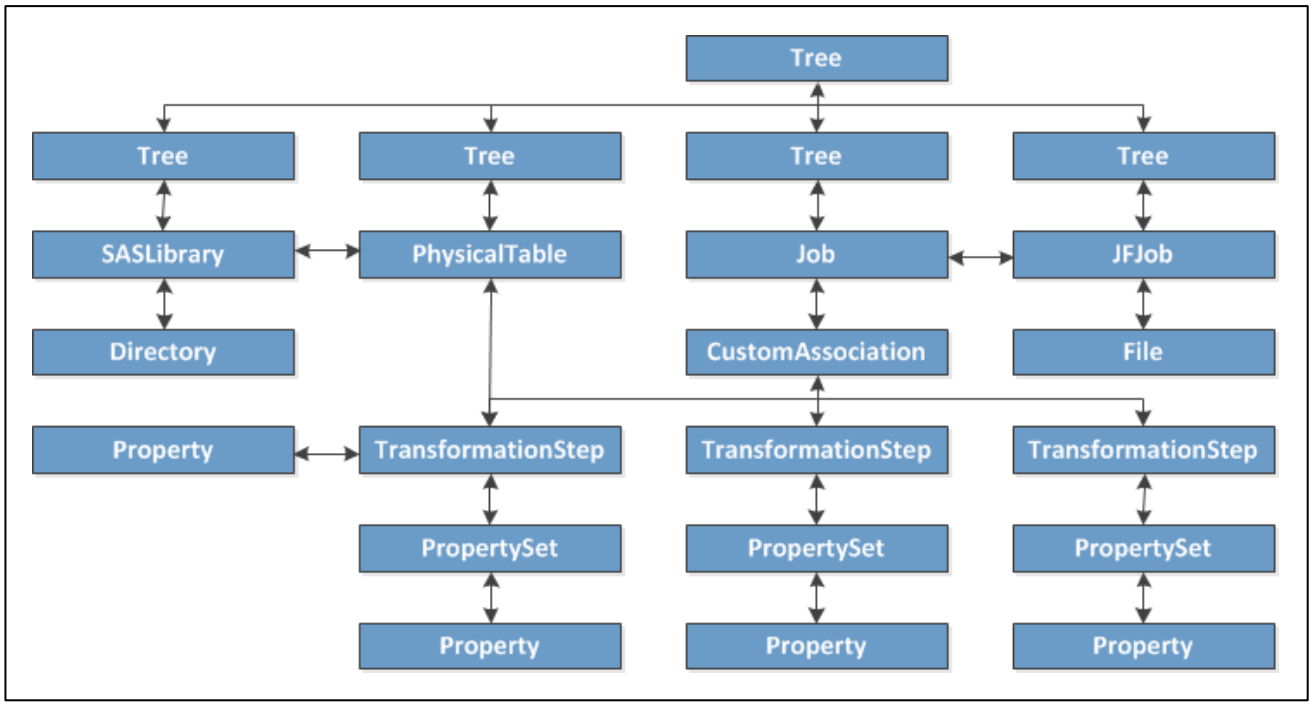

Figure 1. Example of the typical associations between metadata objects

Figure 2 shows an example of the types of attribute you might find for seven of the metadata objects from Figure 1. The attributes each object has depend on their metadata type but there are some attributes which are common across almost all objects.

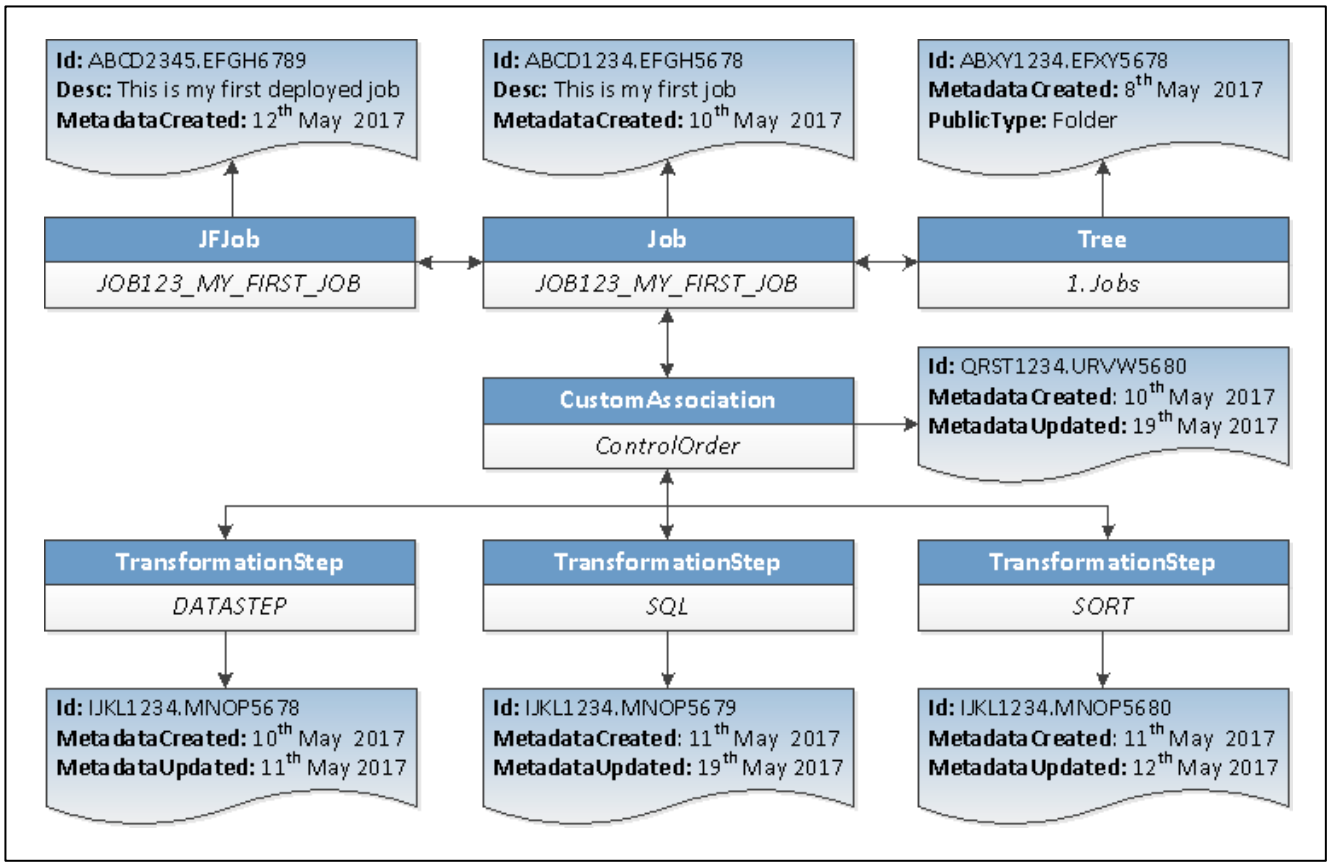

Figure 2. Example of the typical attributes of metadata objects

Metadata – Building Blocks

The third step in metadata management is ensuring you have the necessary tools to extract the relevant information from the metadata. These tools will be referred to as the building blocks and with just five building blocks you can harvest a wealth of information from the metadata. Each building block section includes a snippet of code which will put information to the log. Within the appendices these snippets have been incorporated into macros to provide a more powerful tool kit.

Building Blocks – Types

The first building block in metadata management is listing which types of object exist on the server. Using the METADATA_GETNTYP function within a loop we can get all types. “Job”, “SASLibrary” and “PhysicalTable” are just a few of the over 100 unique object types that exist.

Appendix A is the “metadata_listtyp” macro which uses a modification of this code to write out all of the types of object to a work dataset called “_listtyp_output_”.

```
data _null_;
  length
    type $32
    ;
  rc1=1;
  i=1;
  do until (rc1<=0);
    rc1=metadata_getntyp(i,type);
    if (rc1>0) then do;
      put type=;
    end;
    i=i+1;
  end;
```

```
run;
```

Building Blocks – Objects

The second building block in metadata management is extracting a list of objects for a given type. Using the METADATA_GETNOBJ function within a loop we can get all objects for the given type. In the example below we are extracting all “Job” objects.

Appendix B is the “metadata_listobj” macro which uses a modification of this code to write out all of the URIs for a particular object type to a work dataset called “_listobj_output_”.

```
data _null_;
  length
    uri $64
    ;
  rc1=1;
  i=1;
  do until (rc1<=0);
    rc1=metadata_getnobj("OMSOBJ:JOB?@Id contains '.'",i,uri);
    if (rc1>0) then do;
      put uri=;
    end;
    i=i+1;
  end;
run;
```

Building Block – Attributes

The third building block in metadata management is extracting the attributes of a given object. Using the METADATA_GETNATR function within a loop we can get all attributes for the given object. In the example below we are extracting all attributes for the Job object with ID “ABCD1234.EFGH5678”.

Appendix C is the “metadata_listattr” macro which uses a modification of this code to write out one or more attributes for one or more URIs to a work dataset called “_listattr_output_”.

```
data _null_;
  length
    attr $256
    value $512
    ;
  rc1=1;
  i=1;
  do until (rc1<=0);
    rc1=metadata_getnatr("OMSOBJ:JOB\ABCD1234.EFGH5678",i,attr,value);
    if (rc1>0) then do;
      put attr= value=;
    end;
    i=i+1;
  end;
run;
```

Building Blocks – Associations

The fourth building block in metadata management is extracting the associations of a given object. Using the METADATA_GETNASL function we get the types of association before using the METADATA_GETNASN function to get all associated objects per association type. It is important to use nested loops here to capture all associations. There are several types of object you will typically find associated to a Job; the Control

Order, a Deployed Job, a location in the metadata tree, and multiple property associations which are often grouped into property sets.

Appendix D is the “metadata_listassoc” macro which uses a modification of this code to write out one or more association types for one or more URIs to a work dataset called “_listassoc_output_”.

```
data _null_;
  length
    assoc $32
    assoc_uri $64
    ;
  rc1=1;
  i=1;
  do until(rc1<=0);
    rc1=metadata_getnasl("OMSOBJ:JOB\ABCD1234.EFGH5678",i,assoc);
    rc2=1;
    j=1;
    do until(rc2<=0);
    rc2=metadata_getnasn("OMSOBJ:JOB\ABCD1234.EFGH5678",assoc,j,assoc_uri);
      if (rc2>0) then do;
        put assoc= assoc_uri=;
      end;
      j=j+1;
    end;
    i=i+1;
  end;
run;
```

Building Blocks – Properties

The fifth and final building block in metadata management is extracting the properties of a given object. Using the METADATA_GETNPRP function within a loop we can get all properties for the given object. METADATA_GETNPRP returns the property name and value but doesn’t return the URI so we need to use METADATA_GETPROP with the properties we have found to extract the property URI. In the example below we are extracting all properties for the Job object with ID “ABCD1234.EFGH5678”.

It should be noted that not all objects have properties and sometimes properties are wrapped into property sets and so you would need to look at the properties of a property set rather than the object the property set is associated to in order to get the property information.

Appendix E is the “metadata_listprop” macro which uses a modification of this code to write out one or more properties for one or more URIs to a work dataset called “_listprop_output_”.

```
data _null_;
  length
    prop $256
    value $256
    prop_uri $64
    ;
  rc1=1;
  i=1;
  do until(rc1<=0);
    rc1=metadata_getnprp("OMSOBJ:JOB\ABCD1234.EFGH5678",i,prop,value);
    if (rc1>0) then do;
      rc2=metadata_getprop("OMSOBJ:JOB\ABCD1234.EFGH5678",prop,value,prop_uri);
      if (rc2>=0) then do;
        output;
      end;
    end;
    i=i+1;
```

```
  end;
run;
```

Metadata – Reporting Examples

Once you understand the building blocks and have compiled the associated macros from the appendix you have all the basic tools you need to start reporting on the contents of the metadata layer. However, before you can start generating reports you first need to decide what it is you want to govern and control so you have a framework to report on.

The reporting example sections demonstrate the types of information you can harvest from the metadata and how to do so.

Reporting Example – Jobs with multiple deployed jobs

If you have a policy that each job should only have one corresponding deployed job then you will want to validate this policy is being adhered to. The simplest way to do this is to get a list of jobs, look at their associations to deployed jobs and count for each job how many associations they have to deployed jobs. If it is fine for jobs not to have a deployed job we would define the acceptable number of associated deployed jobs to be either 0 or 1; in which case anything greater than 1 would be a breach.

```
*** Get a list of all job objects. ;
%metadata_listobj(obj=Job);

*** Get the name attribute for each job. ;
%metadata_listattr(omsobj=work._listobj_output_, attr=Name);

*** For each job get all associated deployed jobs. ;
%metadata_listassoc(omsobj=work._listattr_output_, assoc=JFJobs);

proc sql;
create table work.jobs_jfjobs as
  select
    a.uri,
    a.value as name,
    sum(case when b.uri ^= '' then 1 else 0 end) as deployed_job_count
  from work._listattr_output_ as a
  left join work._listassoc_output_ as b
    on a.uri = b.uri
  group by a.uri, a.value
  having deployed_job_count > 1
;quit;
```

Reporting Example – User written code

Within a Job there are lots of ways to inject code. At a node level you can choose between “Automatic”, “User written body” and “All user written” for your code generation but you are actually faced with this same choice at a Job level, allowing the user to override the code produced by the nodes in favour of their own user written code. If you have a policy that user written code is fine at a node level but not at a job level then you will want to validate this policy is being adhered to. In this instance there is a job level attribute called “IsUserDefined” which has a value of 0 when the code generation is “Automatic” and a value of 1 when the code generation is either “User written body” or “All user written”.

```
*** Get a list of all job objects. ;
%metadata_listobj(obj=Job);
```

```
*** Get the name attribute for each job. ;
%metadata_listattr(omsobj=work._listobj_output_, attr=Name);

proc sql;
create table work.jobs_base as
  select
    a.uri,
    a.value as name
  from work._listattr_output_ as a
  order by name
;quit;

*** Get the is user defined attribute for each job. ;
%metadata_listattr(omsobj=work.jobs_base, attr=IsUserDefined);

proc sql;
create table work.jobs_user_defined as
  select
    a.uri,
    a.name,
    b.value as IsUserDefined
  from work.jobs_base as a
  left join work._listattr_output_ as b
    on a.uri = b.uri
  where b.value = '1'
  order by name
;quit;
```

Reporting Example – Libraries referencing the same paths

You could have a policy that each physical path should only be pointed to by at most one library. To validate this you would normally have to look at each libname definition and cross reference to see if any are referencing the same path. Using the metadata we can get a list of all SAS libraries, filter by engine to only keep BASE libraries and then get the paths the libraries are pointing to by finding the “UsingPackages” associations. At this point we just need to count how many times each path is referenced and then filter to keep those referenced more than once.

```
*** Get a list of all SAS library objects. ;
%metadata_listobj(obj=SASLibrary);

*** Get the name attribute for each SAS library. ;
%metadata_listattr(omsobj=work._listobj_output_, attr=Name);

proc sql;
create table work.library_base as
  select
    a.uri,
    a.value as name
  from work._listattr_output_ as a
  order by name
;quit;

*** Get the engine attribute for each SAS library. ;
%metadata_listattr(omsobj=work.library_base, attr=Engine);

*** Filter to only keep BASE libraries. ;
proc sql;
create table work.library_engine as
  select
    a.uri,
    a.name,
    b.value as engine
  from work.library_base as a
  inner join work._listattr_output_ as b
    on a.uri = b.uri
```

```
  where engine = 'BASE'
  order by name
;quit;

*** Get the using packages associations for each SAS library. ;
%metadata_listassoc(omsobj=work.library_engine, assoc=UsingPackages);

*** Get the directory name attribute for each package. ;
%metadata_listattr(omsobj=work._listassoc_output_, uri=assoc_uri, attr=DirectoryName);

*** Merge together the gathered information. ;
proc sql;
create table work.library_path as
  select
    a.uri,
    a.name,
    b.assoc_uri as path_uri,
    c.value as path
  from work.library_engine as a
  left join work._listassoc_output_ as b
    on a.uri = b.uri
  left join work._listattr_output_ as c
    on b.assoc_uri = c.uri
  order by name
;quit;

*** Count how many times each path is referenced. ;
proc sql;
create table work.library_path_count as
  select
    path,
    count(*) as path_count
  from work.library_path
  group by path
;quit;

*** Output the records where the path is referenced more than once. ;
proc sql;
create table work.library_path_duplicates as
  select
    a.uri,
    a.name,
    a.path_uri,
    a.path,
    b.path_count
  from work.library_path as a
  inner join work.library_path_count as b
    on a.path = b.path
  where b.path_count > 1
  order by a.path, a.name
;quit;
```

Metadata – General Solutions

Having covered off a few specific problems and the steps to tackle them, this next section will look at the questions posed in the introduction and the general solution to those problems and problems like them.

General Solution – Are there any user written nodes within a Job?

Usually the validation of how nodes interact takes place within the context of them belonging to a Job and so the start point is always to look at the Job, find the associated nodes and then inspect the nodes for the characteristic you are looking to validate. A reasonable solution to the question of whether there are any user written nodes within a job would be:

(1) Get a list of all the job objects

(2) Get the relevant attributes of the job to help you identify them e.g. name
(3) From the job object you need to find the job activities which have a metadata type of transformation activity.
(4) From the transformation activity you need to find the steps which have a metadata type of transformation step. These are the actual nodes.
(5) For the nodes you then need to get the is user defined attribute and those which have a value of 1 are user defined
(6) To summarise at a job level you would then need to group on job and set a flag or count to identify the number of nodes which use user written code

General Solution – Do multiple metadata table objects reference the same physical table?

Whenever we are looking to determine duplication within the metadata the solution always relies on finding an attribute or multiple attributes that should act as a primary or composite key. Depending on the environment configuration there may need to be some filtering applied first. For example if you have multiple business environments visible to the same user in the metadata, then you may want to filter on a high level folder in the metadata tree. To get the metadata tree path you need to first get the tree the object is associated to and then recursively find the parent tree association for each tree until you've built the full path i.e. at the top level there is no parent tree. A reasonable solution to the question of whether multiple metadata table objects reference the same physical table would be:

(1) Get a list of all the physical table objects
(2) Apply appropriate filtering such as a requirement to belong to a particular branch of the metadata tree.
(3) Get the name and table name attributes for the physical table. The name attribute is the metadata name whereas the table name is the actual physical name of the table
(4) Get the associated metadata library. Assuming you have already implemented the reporting example to ensure no two libraries reference the same path then the composite key you are looking to test will be the metadata library and the physical table name.
(5) Group by metadata library and physical table name and count the number of instances the physical table name occurs within that library.
(6) Any counts greater than one show physical tables which are referenced more than once within a metadata library.

General Solution – Has a particular property been set to the desired value?

Knowing exactly which properties in metadata correspond to particular options in SAS Data Integration Studio can be a little tricky. If you wanted to ensure that all jobs have a particular option set, you first need to know what that corresponds to within the metadata. The easiest way to find this out is to make sure you have saved the job, then take a snapshot of the metadata for that job; this would involve creating datasets that contain the jobs attributes, properties and also the associated property sets and their properties. If you then change a single option and re-save the job you can take another snapshot to see which attribute or property has changed. A reasonable solution to the question of whether a particular property has been set to the desired value would be:

(1) Establish what the property is and at what level the property sits e.g. job, node
(2) Get a listing of the appropriate object e.g. job
(3) Get the relevant attributes that help you identify the object e.g. name

(4) If the property is a direct property of the object then get the metadata property value, otherwise navigate to the correct level by using the metadata associations
(5) Check whether the value matches the expected value

General Solution – Does a flow begin and end with some generic start and stop jobs?

If you have some generic start and stop jobs that always run at the beginning and end of each of your flows to test for trigger files or do housekeeping then you will want to validate these standards are being met. There are two parts to fully solving this problem. First we want to validate that the jobs are actually in the flow, but then second we want to check that the start job is the only job which runs first and the stop job is the only job that runs last. A reasonable solution to the question of whether a flow begins and ends with some generic start and stop job would be:

(1) Get a list of all the flow objects. Due to both deployed jobs and deployed flows having the same object type you first need to get the JFJob objects and you then need to filter on the public type attribute being deployed flow
(2) For each flow you then need to get the transformation activity association which is then associated to transformation steps but in this context the steps are the deployed jobs (nodes within a Job and deployed jobs within a flow both have the same metadata type of transformation step)
(3) At this point we can validate that the appropriate start and stop jobs are included as steps
(4) In order to validate that the start job is the only job which runs first we need to check that it is the only step which does not have any predecessor dependency associations. Predecessor dependencies mean there is another deployed job within the flow that runs before the deployed job you are inspecting. The easiest way to validate there is only one first step is to count the number of predecessor dependencies for each step and check only the start job has zero
(5) In order to validate that the stop job is the only job which runs last we need to check that it is the only step which does not have any successor dependency associations. Successor dependencies mean there is another deployed job within the flow that runs after the deployed job you are inspecting. The easiest way to validate there is only one last step is to count the number of successor dependencies for each step and check only the stop job has zero

Conclusion

This paper provides SAS Data Integration Studio developers with the basic building blocks they need to perform metadata management. The worked examples should help users get up and running with their first metadata reports, while the general solutions should give users an understanding of how much governance and control can be established with the correct framework.

While this paper has covered the detection of problems in the metadata, it hasn't touched on any automated solutions for resolving these problems other than highlighting them. The next stages of metadata management which will require are future paper are:

- Automatically fixing issues that have been detected with metadata commands
- Automated metadata housekeeping
- Creating brand new metadata objects with metadata commands

Appendix A – %metadata_listtyp

This macro does not use any parameters; it always returns all metadata object types in the metadata repository.

```
%macro metadata_listtyp() /
des = 'Use the metadata_getntyp function to get a list of each object type.';

  %*** Get a list of all the possible types of object. ;
  data work._listtyp_output_ (keep=type);
    length
      type $32
      ;
    rc1=1;
    i=1;
    do until (rc1<=0);
      rc1=metadata_getntyp(i,type);
      if (rc1>0) then do;
        output;
      end;
      i=i+1;
    end;
  run;

%mend metadata_listtyp;
```

Appendix B – %metadata_listobj

This macro is controlled by a single parameter.

- obj – must be a valid metadata object type returned by the “metadata_listtyp” macro

```
%macro metadata_listobj(obj=) /
des = 'Use the metadata_getnobj function to get a list of objects for an object type';

  %*** Get the list of objects for a specific object type. ;
  data work._listobj_output_ (keep=uri);
    length
      uri $64
      ;
    rc1=1;
    i=1;
    query=cats('OMSOBJ:',"&obj.",'?@Id contains ',"'",'.',"'");
    do until (rc1<=0);
      rc1=metadata_getnobj(query,i,uri);
      if (rc1>0) then do;
        output;
      end;
      i=i+1;
    end;
  run;

%mend metadata_listobj;
```

Appendix C – %metadata_listattr

This macro is controlled by a series of parameters which give the user a lot of flexibility for extracting attribute information.

- omsobj – must be either a valid URI reference (e.g. the type returned by “metadata_listobj”) or a dataset which contains a list of URIs
- uri – when using a dataset as input where the URI column is not called “uri” then the name of the column which contains the URIs must be specified here
- attr – by default the macro will return all attributes for all URIs, but if only one attribute type is needed it can be specified here and the macro will instead return only this attribute information

```
%macro metadata_listattr(omsobj=, uri=, attr=) /
des = 'Use the metadata_getnatr function to gather attribute information.';

  %*** If the value of omsobj is a dataset that exists then do ... ;
  %if %sysfunc(exist(%superq(omsobj))) %then %do;

    %*** If uri macro variable has not been specified then use a default value of uri. ;
    %if %sysevalf(%superq(uri)=,boolean) = 1 %then %do;
      %let uri=uri;
    %end;

    %*** Create a dataset with the distinct omsobj stored in the uri variable. ;
    proc sql;
    create table work._uri_input_ as
      select distinct
        %superq(uri) as uri length=64
      from %superq(omsobj)
    ;quit;

  %end;
  %*** ... otherwise check if the value begins with OMSOBJ: ... ;
  %else %if %sysfunc(substr(%sysfunc(upcase(%superq(omsobj))),1,%length(OMSOBJ:))) =
OMSOBJ: %then %do;

    %*** Create a single row dataset with the omsobj stored in the uri variable. ;
    data work._uri_input_;
      length
        uri $64.
        ;
      uri = "%superq(omsobj)";
    run;

  %end;
  %*** ... otherwise put an (er)ror to the log. ;
  %else %do;
    %put %str(ER)ROR: An invalid value has been detected as input (%superq(omsobj)). ;
  %end;

  %*** If there is no attr value populated then perform a general attribute query. ;
  %if %sysevalf(%superq(attr)=,boolean) = 1 %then %do;

    data work._listattr_output_ (keep=uri attr value);
      set work._uri_input_;
      length
        attr $256
        value $512
        ;
      rc1=1;
      i=1;
      do until(rc1<=0);
        rc1=metadata_getnatr(uri,i,attr,value);
        if (rc1>0) then do;
          output;
        end;
```

```
        i=i+1;
      end;
    run;

  %end;
  %*** If there is an attr value populated then perform a specific attribute query. ;
  %else %do;

    data work._listattr_output_ (keep=uri attr value);
      set work._uri_input_;
      length
        attr $256
        value $512
        ;
      attr = "%superq(attr)";
      rc1=metadata_getattr(uri,attr,value);
      if (rc1>=0) then do;
        output;
      end;
    run;

  %end;

  %*** Delete the temp table(s). ;
  proc datasets lib=work nolist;
    delete
      _uri_input_
      ;
    run;
  quit;

%mend metadata_listattr;
```

Appendix D – %metadata_listassoc

This macro is controlled by a series of parameters which give the user a lot of flexibility for extracting association information.

- omsobj – must be either a valid URI reference (e.g. the type returned by “metadata_listobj”) or a dataset which contains a list of URIs
- uri – when using a dataset as input where the URI column is not called “uri” then the name of the column which contains the URIs must be specified here
- assoc – by default the macro will return all associations for all URIs, but if only one association type is needed it can be specified here and the macro will instead return only this association information

```
%macro metadata_listassoc(omsobj=, uri=, assoc=) /
des = 'Use the metadata_getnasn function to gather association information.';

  %*** If the value of omsobj is a datset that exists then do ... ;
  %if %sysfunc(exist(%superq(omsobj))) %then %do;

    %*** If uri macro variable has not been specified then use a default value of uri. ;
    %if %sysevalf(%superq(uri)=,boolean) = 1 %then %do;
      %let uri=uri;
    %end;

    %*** Create a dataset with the distinct omsobj stored in the uri variable. ;
    proc sql;
    create table work._uri_input_ as
      select distinct
        %superq(uri) as uri length=64
      from %superq(omsobj)
    ;quit;

  %end;
  %*** ... otherwise check if the value begins with OMSOBJ: ;
  %else %if %sysfunc(substr(%sysfunc(upcase(%superq(omsobj))),1,%length(OMSOBJ:))) =
OMSOBJ: %then %do;

    %*** Create a single row dataset with the omsobj stored in the uri variable. ;
    data work._uri_input_;
      length
        uri $64.
        ;
      uri = "%superq(omsobj)";
    run;

  %end;
  %*** ... otherwise put an (er)ror to the log. ;
  %else %do;
    %put %str(ER)ROR: An invalid value has been detected as input (%superq(omsobj)). ;
  %end;

  %*** If there is no assoc value populated then perform a general association query. ;
  %if %sysevalf(%superq(assoc)=,boolean) = 1 %then %do;

    data work._listassoc_output_ (keep=uri assoc assoc_uri);
      set work._uri_input_;
      length
        assoc $32
        assoc_uri $64
        ;
      rc1=1;
      i=1;
      do until(rc1<=0);
        rc1=metadata_getnasl(uri,i,assoc);
        rc2=1;
        j=1;
        do until(rc2<=0);
```

```
          rc2=metadata_getnasn(uri,assoc,j,assoc_uri);
          if (rc2>0) then do;
            output;
          end;
          j=j+1;
        end;
        i=i+1;
      end;
    run;

  %end;
  %*** If there is an assoc value populated then perform a specific association query. ;
  %else %do;

    data work._listassoc_output_ (keep=uri assoc assoc_uri);
      set work._uri_input_;
      length
        assoc $32
        assoc_uri $64
        ;
      assoc = "%superq(assoc)";
      rc1=1;
      i=1;
      do until(rc1<=0);
        rc1=metadata_getnasn(uri,assoc,i,assoc_uri);
        if (rc1>0) then do;
          output;
        end;
        i=i+1;
      end;
    run;

  %end;

  %*** Delete the temp table(s). ;
  proc datasets lib=work nolist;
    delete
      _uri_input_
      ;
    run;
  quit;

%mend metadata_listassoc;
```

Appendix E – %metadata_listprop

This macro is controlled by a series of parameters which give the user a lot of flexibility for extracting property information.

- omsobj – must be either a valid URI reference (e.g. the type returned by “metadata_listobj”) or a dataset which contains a list of URIs
- uri – when using a dataset as input where the URI column is not called “uri” then the name of the column which contains the URIs must be specified here
- assoc – by default the macro will return all properties for all URIs, but if only one property type is needed it can be specified here and the macro will instead return only this property information

```
%macro metadata_listprop(omsobj=, uri=, prop=) /
des = 'Use the metadata_getnasn function to gather association information.';

  %*** If the value of omsobj is a datset that exists then do ... ;
  %if %sysfunc(exist(%superq(omsobj))) %then %do;

    %*** If uri macro variable has not been specified then use a default value of uri. ;
    %if %sysevalf(%superq(uri)=,boolean) = 1 %then %do;
      %let uri=uri;
    %end;

    %*** Create a dataset with the distinct OMSOBJ stored in the URI variable. ;
    proc sql;
    create table work._uri_input_ as
      select distinct
        %superq(uri) as uri length=64
      from %superq(omsobj)
    ;quit;

  %end;
  %*** ... otherwise check if the value begins with OMSOBJ: ;
  %else %if %sysfunc(substr(%sysfunc(upcase(%superq(omsobj))),1,%length(OMSOBJ:))) =
OMSOBJ: %then %do;

    %*** Create a single row dataset with the OMSOBJ stored in the URI variable. ;
    data work._uri_input_;
      length
        uri $64.
        ;
      uri = "%superq(omsobj)";
    run;

  %end;
  %*** ... otherwise put an (er)ror to the log. ;
  %else %do;
    %put %str(ER)ROR: An invalid value has been detected as input (%superq(omsobj)). ;
  %end;

  %*** If there is no prop value populated then perform a general property query. ;
  %if %sysevalf(%superq(prop)=,boolean) = 1 %then %do;

    data work._listprop_output_ (keep=uri prop value prop_uri);
      set work._uri_input_;
      length
        prop $256
        value $256
        prop_uri $64
        ;
      rc1=1;
      i=1;
      do until(rc1<=0);
        rc1=metadata_getnprp(uri,i,prop,value);
        if (rc1>0) then do;
          rc2=metadata_getprop(uri,prop,value,prop_uri);
```

```
          if (rc2>=0) then do;
            output;
          end;
        end;
        i=i+1;
      end;
    run;

  %end;
  %*** If there is a prop value populated then perform a specific property query. ;
  %else %do;

    data work._listprop_output_ (keep=uri prop value prop_uri);
      set work._uri_input_;
      length
        prop $256
        value $256
        prop_uri $64
        ;
      prop = "%superq(prop)";
      rc1=metadata_getprop(uri,prop,value,prop_uri);
      if (rc1>=0) then do;
        output;
      end;
    run;

  %end;

  %*** Delete the temp table(s). ;
  proc datasets lib=work nolist;
    delete
      _uri_input_
      ;
    run;
  quit;

%mend metadata_listprop;
```

References

"http://documentation.sas.com/?docsetId=lrmeta&docsetTarget=p0o5rdel8tgzgsn1iqhdwybgui7k.htm&docsetVersion=9.4&locale=en"

Contact Information

Your comments and questions are valued and encouraged. Contact the authors at:

Amit Patel, Email: amit.xc.patel@barclayscorp.com
Amit Patel has been developing solutions across a range of SAS products since 2011 and has worked as a SAS technical lead within several organisations including two of the big four banks in the UK.

Lewis Mitchell, Email: lewis.mitchell@barclayscorp.com
Lewis Mitchell has been developing solutions across a range of SAS products since 2009 and has worked as a SAS technical lead within several organisations including three of the big four banks in the UK.